МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ УКРАИНЫ

«ПРОГРАММИРОВАНИЕ НА АССЕМБЛЕРЕ»

методические указания
к лабораторным работам
по курсу «Основы микропроцессорной техники»
для студентов специальностей 7.091301
«Информационно-измерительные системы», 7.091302
«Метрология и измерительная техника»
дневной и заочной форм обучения

Утверждено
редакционно-издательским
советом университета,
протокол №

Харьков НТУ «ХПИ» 2007

«Програмування на асемблері», методичні вказівки до лабораторних робіт з курсу «Основи мікропроцесорної техніки», для студентів спеціальностей 7.091301 «Інформаційно-вимірювальні системи», 7.091302 «Метрологія та вимірювальна техніка» денної та заочної форм навчання / Уклад. О.І. Овчаренко, В.В. Лисенко, Р.П. Мигущенко, Л.О. Медведєва, О.Ю. Кропачек, М.В. Шапіро – Харків: НТУ «ХПІ», 2007.– 40 с.– Рос. мовою

Укладачі: О.І. Овчаренко,
В.В. Лисенко,
Р.П. Мигущенко,
Л.О. Медведєва,
О.Ю. Кропачек,
М.В. Шапіро

Рецензент В.І. Дякін

Кафедра інформаційно-вимірювальних технологій та систем

Лабораторная работа № 1

ИЗУЧЕНИЕ СТРУКТУРЫ И ПРАВИЛ ПОЛЬЗОВАНИЯ ДИРЕКТИВАМИ СРЕДЫ IS580

Цель работы: изучить порядок запуска программы IS580; изучить директивы IS580; изучить правила пользования директивами IS580; приобрести практические навыки в использовании директив IS580.

Опыт 1. Запуск программы IS580

Запуск осуществляется путем входа в папку IS580 ПК и запуска программы start нажатием клавиши «Enter» клавиатуры.

Порядок выполнения работы

1. Включить питание ПК;
2. Войти в папку IS580;
3. Запустить программу start нажатием клавиши «Enter» клавиатуры ;
4. В ответ на это на экране видеотерминала должно появится окно с системой IS580. После этого можно приступать к работе с программой IS580;
5. При отсутствии выхода в систему IS580 (п.4) – перезагрузить ПК и повторить опыт 1, начиная с п.2. Если присутствует сбой и при повторном проведении опыта, необходимо обратится к преподавателю.

Опыт 2. Изучение структуры программы IS580

Программа IS580 является командно-управляемым операционным супервизором, который принимает команды оператора и управляет работой вычислительной системы. IS580 связан с оператором при помощи консоли (устройство отображения информации на ЭЛТ и клавиатуре). Диалог между оператором и IS580 осуществляется при помощи команд (директив), набираемых и вводимых с клавиатуры, и ответных сообщений в виде блок-схем, показанных на рисунке 1.

Из указанной схемы следует:

1. После запуска программы отладчика системы IS580 пользователь попадает в блок-меню выбора директив;
2. Набором и вводом пользователь выполняет следующие функции:

– отображение на экране видеотерминала содержимого участка памяти (директива D);

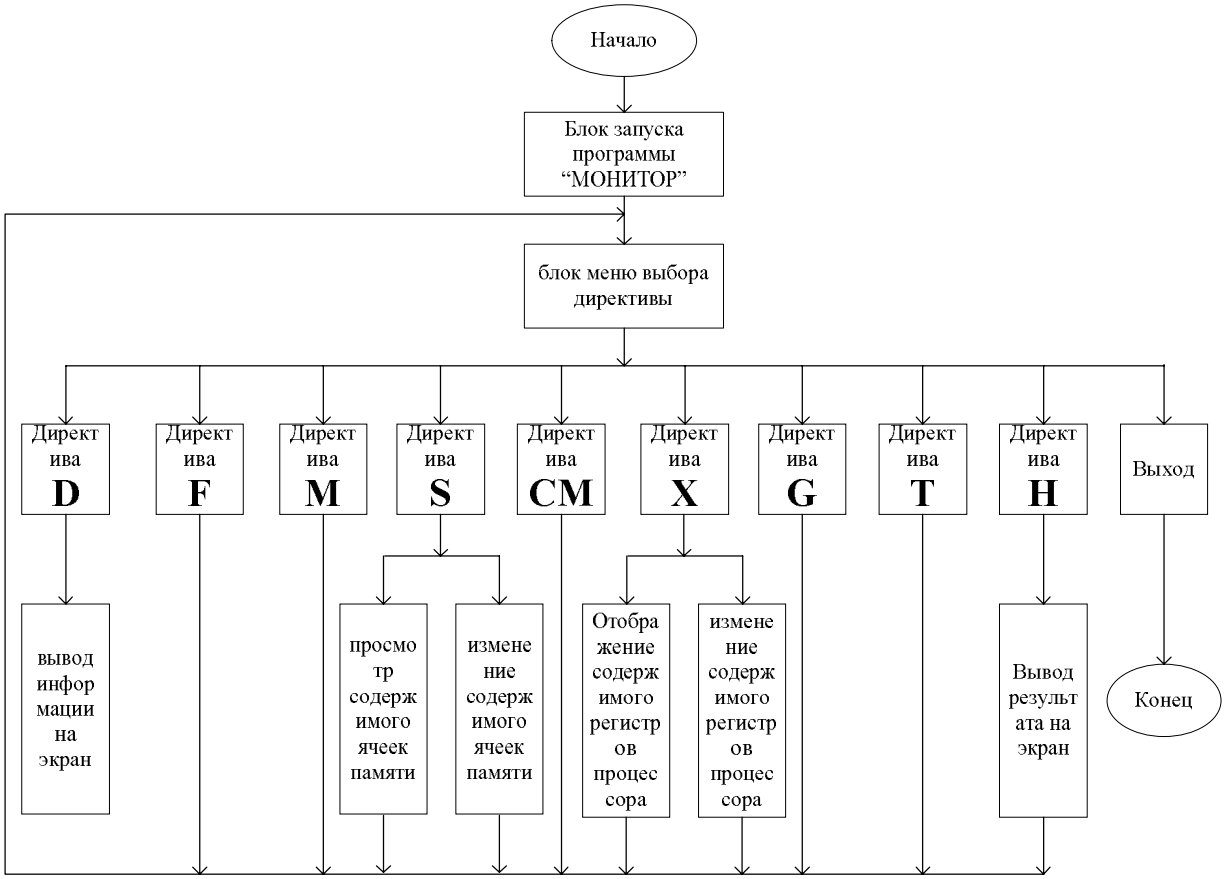


Рисунок 1 – Блок-схема взаимодействия программы IS580 и оператора.

– заполнение области памяти константой (директива F);

– пересылка содержимого памяти из одной области в другую (директива M);

– посредством директивы S выполняется:

а) просмотр содержимого ячеек памяти;

б) изменение содержимого ячеек памяти.

– сравнение содержимого двух областей памяти (директива C);

– отображение на экране видеотерминала содержимого регистров процессора и изменения их (директива R);

– выполнение программы (директива G).

3. После выполнения одной из директив, необходимой пользователю, идет автоматический возврат в блок-меню выбора директивы;

4. Для выхода из программы IS580 необходимо использовать нажатие клавиши F10 и Y.

Формат директив приведен в таблице 1.

Таблица 1 – Формат директив

№ п/п	Формат директив	Описание директив
1	D ADR1, ADR2 ВК	Отобразить содержание области памяти
2	F ADR1, ADR2, Y ВК	Заполнить область памяти константой Y, Y ∈ (00...FF)
3	M ADR1, ADR2, ADR ВК	Переслать содержимое памяти из одной области в другую
4	S ADR1 ВК	Изменить (просмотр) содержимого ячеек памяти начиная с адреса ADR1
5	C ADR1, ADR2, ADR3 ВК	Сравнить содержимое двух областей памяти
6	R ВК	Отобразить (изменить) содержимое регистров процессора.
7	G ВК	Запустить ассемблерную программу

Примечание: 1. ADR1 – шестнадцатеричный адрес начала области памяти;
ADR2 – адрес конца области памяти;
ADR3 – адрес начала второй области памяти;
2. Максимальные значения адресов памяти определяются объёмом ОЗУ, для лабораторных работ составляет FFFFH.

Порядок выполнения опыта

1. Набрать на клавиатуре директиву D, в соответствии с таблицей 1. Адреса ADR1 и ADR2, соответствующие границам области памяти, задает преподаватель. Ввести директиву нажатием клавиши «Enter» (ВК). Занести адреса и данные отображаемой области памяти в таблицу 2;

Таблица 2 – Адреса и данные отображаемой области памяти

Адрес	Данные

2. Набрать на клавиатуре директиву F, в соответствии с таблицей 1. Адреса ADR1, ADR2 и значение константы Y задает преподаватель. Ввести директиву нажатием клавиши ВК. Директивой D отобразить

содержимое области памяти, в которую записывалась константа Y, и проверить правильность выполнения директивы;

3. Набрать на клавиатуре директиву M, в соответствии с таблицей 1. Значения ADR1, ADR2 взять из п.1 данного опыта, значение ADR3 приравнять значению ADR1 и п.2 данного опыта. Ввести директиву нажатием клавиши ВК. Директивой D отобразить содержимое области памяти, начиная с ADR3 и длиной (ADR2-ADR1)H.

Сравнить полученную информацию с данными таблицы 2;

4. Набрать на клавиатуре директиву S, в соответствии с таблицей 1. Значение ADR1 взять из п.1 данного опыта, нажимая несколько раз клавишу ВК, просмотреть содержимое ячеек памяти и сравнить их с данными таблицы 2. Сообщения о данных в памяти разделяются символом "=".

5. Изменить содержимое памяти с начальным адресом ADR1 из п.4 с помощью директивы S. Для этого набрать директиву S, нажать клавишу ВК, набрать новое содержимое ячейки и ввести его нажатием клавиши ВК. В качестве нового содержимого ячеек памяти взять однобайтное шестнадцатеричное число. Для выхода из этого режима необходимо последовательно нажать: "." и ВК;

6. Набрать на клавиатуре и ввести директиву R. При этом на экране появится сообщение о состоянии регистров процессора.

7. Изменить содержимое регистров микропроцессора. Для этой цели необходимо подсветить нужный регистр и ввести его новое значение с клавиатуры. Здесь R – величина переменная. В качестве R используются A, B, C, D, E, H, L, M, P, S. Наименования регистров, содержимое которых нужно изменить, и данные, которые следует занести в эти регистры, задает преподаватель.

Содержание отчета

В отчете по лабораторной работе необходимо привести директивы среды IS580 и их описание, наименование регистров Intel 8080 и их обозначения.

Контрольные вопросы

1. Что такое IS580?
2. Для чего предназначена IS580?
3. Какова структура среды IS580?
4. Как осуществляют запуск среды IS580?
5. Для чего служат директивы D, F, M, S, C, R, I, N, A, G?
6. Каков формат директив D, F, M, S, C, R, I, N, A, G?

Список литературы

1. Монитор. Руководство программиста 1582.02087-1
2. Григорьев В. П. Программное обеспечение микропроцессорных систем. М.: Энергоиздат, 1983. – С. 5 – 80
2. Коган Б. М., Сташин В. В. Микропроцессоры в цифровых сигналах. – М.: Энергия, 1979. – С. 26 – 33

Лабораторная работа № 2

РУЧНАЯ ТРАНСЛЯЦИЯ, ВВОД И ИСПОЛНЕНИЕ МАШИННЫХ КОМАНД

Цель работы: изучить систему команд микропроцессора Intel 8080; приобрести навыки ручной трансляции программ, написанных на языке Ассемблер микропроцессора Intel 8080; освоить процедуры ввода и выполнения основных машинных команд микропроцессора Intel 8080.

Опыт 1. Ручная трансляция программ написанных на языке Ассемблер

Совокупность машинных команд представляет собой базовый микропроцессорный язык. Для облегчения разработки программного обеспечения на основе базового процессорного языка создаются языки программирования более высокого уровня. Программы микропроцессорных систем должны выполнятся в минимальное время,

занимать минимальный объем памяти и легко изменяться при необходимости. Всем этим условиям отвечает программирование на языке Ассемблер, оперирующем с мнемокодами и символьными адресами. Команды Ассемблера позволяют существенно упростить составление, чтение и отладку программ микропроцессорной системы. Каждая машинная команда обозначается символом, представляющим собой сокращенную форму полной записи наименования данной команды на алгоритмическом языке. Символ кодирования названия и содержания команд запоминается и воспринимается намного легче, чем двоичные или шестнадцатеричные коды.

При использовании Ассемблера необходимо учитывать специфические особенности конкретного микропроцессора, т.к. этот язык относится к машинно-ориентированным языкам программирования. Перевод программы, написанной на Ассемблере, в машинные коды называется трансляцией. Процесс трансляции сложных программ выполняется с помощью ЭВМ. Простые программы могут транслироваться вручную с привлечением таблиц. Команды Ассемблера микропроцессора Intel 8080 классифицируются следующим образом:

– команды пересылки;

– арифметические команды;

– логические команды;

– команды управления программой;

– специальные команды.

Ассемблер микропроцессора Intel 8080 имеет возможность работы с однобайтными и двухбайтными числами. Смысл трансляции заключается в переводе ассемблерной программы, выполненной в мнемокодах, в шестнадцатеричные коды.

Пример ручной трансляции программы:

Дан массив, состоящий из десяти чисел, расположенный в области ОЗУ начиная с адреса 1000Н. Переслать массив в область ОЗУ, начиная с адреса 1100Н. Программу пересылки расположить в памяти с адреса 0500Н. Блок-схема алгоритма выполнения данной программы приведена на рисунке 1.

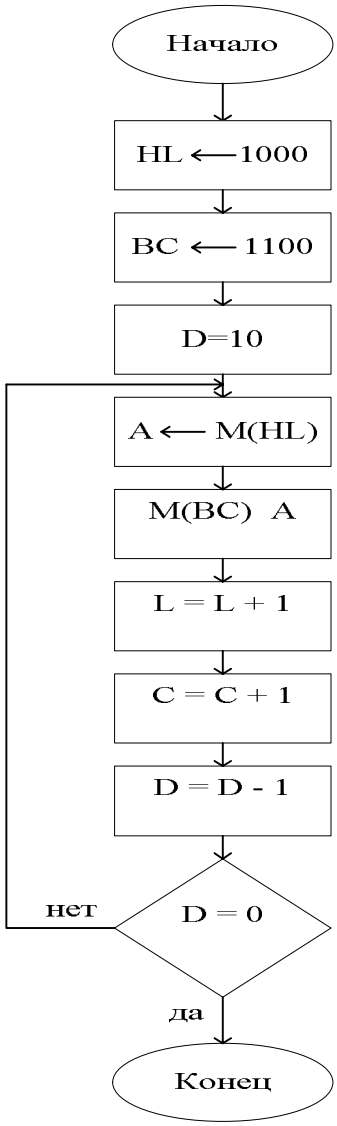


Рисунок 1 – Блок-схема программы

Программа

Адрес	Команда	Двоичный код	Код	Комментарии
0500	LXI HL	00100001	21	Загрузка регистровой пары HL
0501			00	адресом первого массива
0502			12	
0503	LXI BC	00000001	01	Загрузка регистровой пары BC
0504			00	адресом второго массива
0505			11	
0506	MVI D	00010110	16	Загрузка регистра D числом,
0507			0A	характеризующим величину массива
0508	MOV A, M(HL)	01111110	7E	Пересылка содержимого памяти адресуемой регистровой парой HL в аккумулятор
0509	STAX M(BC)	00000010	02	Пересылка содержимого аккумулятора в ячейку памяти, адресуемой регистровой парой BC
050A	INR L	00101100	2C	Увеличение содержимого регистров L

050B	INR C	00001100	0C	и C на единицу
050C	DCR D	000101010	15	Уменьшение содержимого регистра D на единицу
050D	JNZ		C2	Сравнение содержимого регистра D с
050E			08	нулем и переход по
050F			05	соответствующему адресу
0501	HLT		76	Останов

Порядок выполнения опыта

1. Получить от преподавателя задание (в соответствии с таблицей 1), значения адресов и данных;

2. Провести ручную трансляцию программы в машинные коды;

3. Транслированную программу показать преподавателю.

Таблица 1 – Варианты заданий для составления программы

№	Задание
1	Определить максимальный элемент массива
2	Определить минимальный элемент массива
3	Определить количество элементов массива больших 0F H
4	Определить количество элементов массива меньших 0F H
5	Определить младший байт суммы элементов массива больших 0F H
6	Определить младший байт суммы элементов массива меньших 0F H

Опыт 2. Изучение процедур ввода и выполнения ассемблерных программ

Программа на ассемблере должна быть введена в память ПК в данной лабораторной работе с использованием встроенного редактора. Выход в редактор осуществляется при помощи функциональной клавиши F5. Сама программа вводится с клавиатуры. Прогон программы осуществляется в пошаговом режиме с применением нажатий клавиши F7.

Порядок выполнения опыта

1. Ввести программу опыта 1 в память. Для этого необходимо войти в редактор по клавише F5. Далее оттранслировать программу нажав клавишу F9. Обязательно запомнить начальный адрес программы. Вернуться в редактор по «Esc»;

2. Войти в отладчик по клавише «F8». Загрузить транслированную программу с расширением .hex с использованием «F3»;

3. Отобразить содержимое регистров микропроцессора директивой R и результат занести в первую строку таблицы 2;

4. Директивой G назначить начальный адрес выполняемой программы. Результат занести во вторую строку таблицы 2;

5. Провести трассировку выполнения программы, последовательно применяя директиву F7. Результаты трассировки записать в таблицу 2, начиная с третьей строки;

6. По окончанию трассировки результат показать преподавателю.

Таблица 2 – Результаты трассировки программы

№ строки	Адрес, команда, код	Состояния регистров										
		A	B	C	D	E	F	H	L	M	P	S
1	Состояния регистров											
2	Исходное состояние											
3												

Опыт 3. Изучение процедуры автоматического выполнения ассемблерной программы

Отладка и выполнение ассемблерной программы в пошаговом режиме не всегда удобна. В частности большие неудобства возникают при прогоне больших программ, циклических программ, программ, состоящих из множества коротких и отлаженных программ. В этих случаях целесообразно использовать автоматическое выполнение ассемблерных программ.

Порядок выполнения опыта

1. Отобразить на экране видеотерминала в редакторе состояние программы, введенной в опыте 2.

2. Выполнить пункты 1-4 опыта 2:

3. Провести трассировку выполнения программы применяя директиву F9.

Содержание отчета

Отчет по лабораторной работе представляет собой заполненную таблицу 2.

Контрольные вопросы

1. Что такое трансляция, как она осуществляется?

2. Перечислите типы команд микропроцессора Intel 8080.

3. Каково должно быть содержимое программного счетчика к моменту запуска программы?

4. Как изменяется содержимое программного счетчика при пошаговом выполнении программы?

5. Как организовать пересылку в микропроцессор данных длиной в два байта?

Список литературы

1. Григорьев В.А. Программное обеспечение микропроцессорных систем. М.: Энергоиздат, 1983. – С. 5 – 90.

2. Качян Б.М., Сташин В.В. Микропроцессоры в цифровых системах. М.: Энергия, 1979. – С . 26-3353 – 84.

3. Справочник по цифровой вычислительной технике. /Под ред. Б.Н. Малиновского. К.: Техника, 1981. – С. 22-24.

4. Монитор. Руководство программиста. 1582.02087-01.НПО САУ.

Лабораторная работа №3

ИЗУЧЕНИЕ ПРОЦЕДУР РАБОТЫ СО СТЕКОМ

Цель работы: изучение группы команд микропроцессора Intel 8080, относящихся к операциям со стеком; изучение принципов и практических приемов применения этих команд.

ОСНОВНЫЕ ТЕОРЕТИЧЕСКИЕ ПОЛОЖЕНИЯ

Стек – особый вид оперативной памяти, отличающийся последовательным способом организации доступа к ней.

Стек – это область памяти, в которой временно сохраняется информация, необходимая для осуществления возврата в программу после выполнения подпрограммы. Стек также используется для временного

сохранения любых данных при нехватке внутренних регистров МП.

Стек играет огромную роль при обработке прерываний. Реагируя на внешние или внутренние прерывания, МП должен переключиться с текущей программы на другую программу, называемую обработчиком прерывания. Однако после обслуживания прерываний необходимо продолжить выполнения основной программы, для которой в стеке временно сохраняется содержимое используемых обработчиком прерывания регистров МП на время его работы.

Состояние стека отображается во внутреннем 16-битном регистре SP – указателе стека, который содержит адрес вершины стека.

В типовых МП фактическое положение стека в ОЗУ определяется программистом. До использования стека необходимо инициализировать SP командой LXI SP на максимальный адрес области (вершину), выделяемой для стека.

Стек функционирует как память с последовательным доступом по типу данные, поступившие последними, извлекаются первыми (тип LIFO от LAST IN - FIRST OUT – последний входит - первый выходит или FILO от FIRST IN - LAST OUT – первый входит - последний выходит).

Содержимое любого регистра МП можно поместить в стек с помощью команды PUSH, а из стека можно извлечь командой POP последние включенные в него данные регистра МП. Обе стековые операции выполняются с 16-битовым словами, т.е. в стек можно включать только содержимое регистровых пар и извлекать из стека можно только регистровые пары.

Извлечение из стека не разрушает в памяти считываемые данные и, произведя специальными командами декремент SP на 2, можно вновь извлечь из стека ранее считываемые данные. При программировании надо следить за использованием стека, в частности, каждой команде PUSH должна соответствовать команда POP.

В Intel 8080 есть специфическая 1-байтная команда XTHG, которая производит обмен содержимого регистров H, L и двух верхних ячеек стека, т.е. последних загруженных в стек данных, при этом значение SP не изменяется.

К особенностям микропроцессора Intel 8080 относится однобайтная команда SPHL передачи 16-битных данных регистра HL в указатель стека.

Следует отметить, что команда CALL сочетает функции операций перехода и загрузки в стек. Адрес перехода к подпрограмме зафиксирован во втором и третьем байте команды CALL, а адрес следующей команды за CALL отправляется в стек. После завершения выполнения подпрограммы команда RET извлекает из стека адрес следующей команды основной программы и загружает его в счетчик команд PC.

Основным достоинством стека является то, что можно заносить в него данные (увеличивать емкость стека), не разрушая структуру уже записанных в нем данных. Если же данные запоминаются в ячейках памяти или в регистре, то теряется предыдущее содержимое этого участка памяти. Кроме того, МП может быстро передавать данные в стек и из стека, т.к. адрес содержится в указателе стека, а не является частью команды. Формат команд операций со стеком очень короткий.

Однако микропроцессор Intel 8080 не имеет никаких средств аппаратного контроля за поведением стека. Ошибки при работе со стеком приводят к двум ситуациям:

1. Переполнение стека – указатель стека достиг верхней границы области стека и делается попытка произвести дополнительное включение в стек;

2. Антипереполнение стека – указатель стека находится на нижней границе области стека и делается еще одна попытка извлечения из стека.

Возникновение любой из этих ситуаций ведет к тому, что поведение системы становится непредсказуемым.

Опыт 1. Изучение группы команд записи в стек

Микропроцессор Intel 8080 имеет в своей системе команд четыре команды записи в стек как показано в таблице 1.

Все команды таблицы 1 однотипны и осуществляют пересылку двухбайтного слова из регистровых пар микропроцессора в ОЗУ, адресуемое при этих операциях указателем стека SP. Специфической парой регистров является аккумулятор A и регистр признаков F, совместное содержимое которых называется словом состояния программы (PSW).

Таблица 1 – Команды записи в стек

№	Мнемокод команд	Код команды	Выполняемые операции
1	PUSH BC	C5H	M<(sp)-1> ← (B); M<(sp)-2> ← (C); SP ← (SP)-2
2	PUSH DE	D5H	M<(sp)-1> ← (D); M<(sp)-2> ← (E); SP← (SP)-2
3	PUSH HL	E5H	M<(sp)-1> ← (H); M<(sp)-2> ← (L); SP← (SP)-2
4	PUSH PSW	F5H	M<(sp)-1> ← (A); M<(sp)-2> ← (F); SP← (SP)-2

Примечания:

1. В таблице 1 и далее запись (имя регистра) означает содержимое регистра с указанным именем.

Пример: (F) – содержимое регистра признаков F.

2. В таблице 1 и далее запись < > означает адрес.

Пример: M <(sp)-1> – ячейка памяти с адресом, равным исходному SP, уменьшенному на единицу.

Важно понять и запомнить основные механизмы записи в стек:

– старший байт слова (соответственно содержимое регистров B, D, H, A для команд таблицы 1) пересылается в ОЗУ по старшему адресу <(sp)-1>, где (sp) – содержимое указателя стека SP до выполнения команды;

– младший байт слова (соответственно содержимое регистров C, E, L, F для команд таблицы 1) пересылается в ОЗУ по младшему адресу <(sp)-2>, где (sp) – содержимое указателя стека SP до выполнения команды;

– по окончании выполнения команды содержимое указателя стека увеличивается на 2 по отношению к исходному содержимому.

Графическая интерпретация функционирования команд PUSH представлена на рисунке 1 (на примере команды PUSH BC).

Как видно из рисунка 1 и рисунка 2 работа стека при записи напоминает укладку листов в пачку. При этом листы (байты) как бы укладываются парами поверх пачки, а указатель стека хранит номер (адрес) последнего из уложенных листов (байтов).

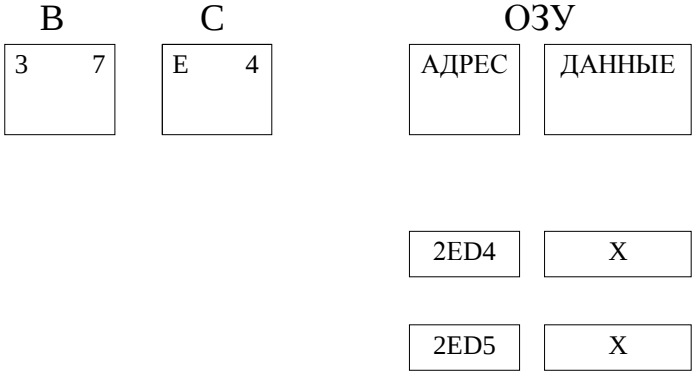


Рисунок 1. – Состояние МП и ОЗУ до выполнения команды PUSH BC:
X – произвольное содержимое

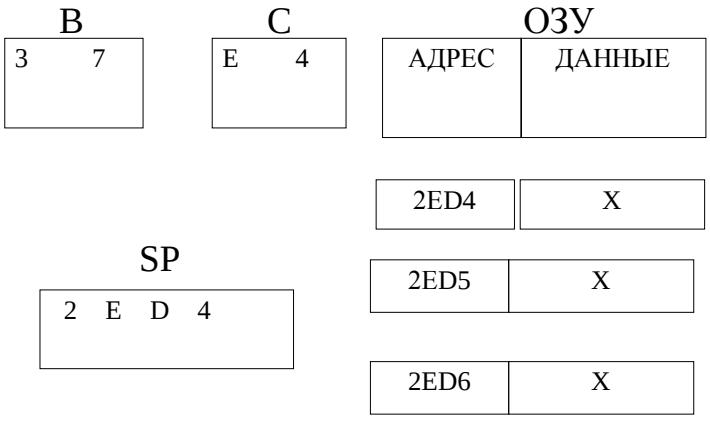


Рисунок 2. – Состояние МП и ОЗУ после выполнения команды PUSH BC:
X – произвольное содержимое

Порядок выполнения опыта

1. Включить рабочее место и запустить программу IS580;

2. Выполнить директиву R IS580. Обратить внимание на то, что всегда после запуска программы IS580 (SP) = 0100 H, (A) = 00H, (B) = 00H, (C) = 00 H, (D) = 00 H, (E) = 00 H, (F) = -----, (H) = 00 H, (L) = 00 H;

3. Директивой S ввести программу записи в стек содержимого регистров МП;

АДРЕС	ДАННЫЕ	
1000	C5	PUSH BC
1001	D5	PUSH DE

1002	E5	PUSH HL
1003	F5	PUSH PSW

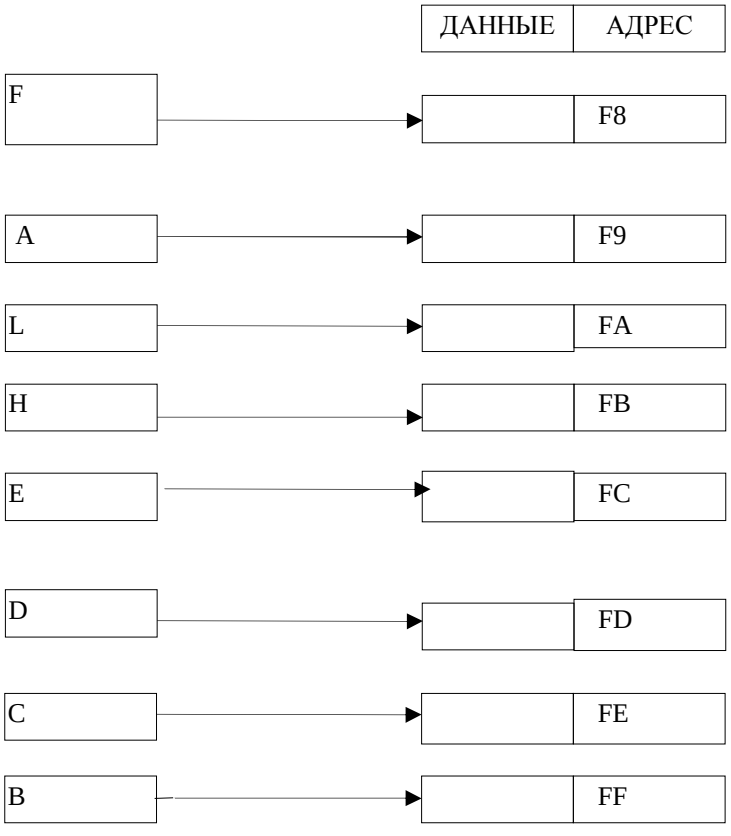
4. Директивой R установить содержимое программного счетчика равным 1000H и нулевое содержимое ячеек памяти по адресам 100H...108H;

5. Выполнить одну команду программы п.3 в пошаговом режиме. Убедиться, что содержимое указателя стека уменьшилось на 2;

6. Выполнить п.5 еще три раза для полного прогона программы п.4;

9. Отобразить содержимое восьми ячеек ОЗУ, начиная с адреса 100H. Убедиться, что результат работы программы п.3 соответствует таблице 2.

Таблица 2 – Результат работы программы записи содержимого регистров МП в стек



Опыт 2. Изучение группы команд чтения из стека

Микропроцессор Intel 8080 имеет в своей системе команд четыре команды чтения из стека (таблица 3).

Таблица 3 – Команды чтения из стека

№	Мнемокод команд	Код команды	Выполняемые операции
1	POP BC	C1H	B ← (M<(sp)+1>); C ← (M<(sp)>); SP ← (SP)+2
2	POP DE	D1H	D ← (M<(sp)+1>); E ← (M<(sp)>); SP ← (SP)+2
3	POP HL	E1H	H ← (M<(sp)+1>); L ← (M<(sp)>); SP ← (SP)+2
4	POP PSW	F1H	A ← (M<(sp)+1>); F ← (M<(sp)>); SP ← (SP)+2

Все команды таблицы 3 однотипны и осуществляют пересылку двухбайтного слова из ОЗУ в регистровые пары МП. Механизмы чтения из стека таковы:

– старший байт слова в ОЗУ, расположенный по старшему адресу <(SP) + 1 >, передается в регистры B, D, H, A (в соответствии с командами таблицы 3);

– младший байт слова в ОЗУ, расположенный по младшему адресу <(SP)>, передается в регистры C, E, L, F (в соответствии с командами таблицы 3);

– по окончании выполнения команды содержимое указателя стека увеличивается на 2 по отношению к исходному состоянию.

Графическая интерпретация функционирования команд POP представлена на рисунке 3 и рисунке 4 (на примере команды POP PSW).

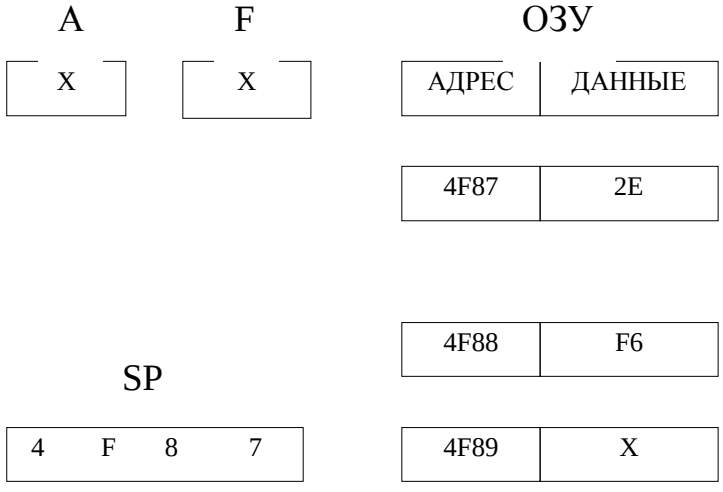


Рисунок 3. – Состояние МП и ОЗУ до выполнения команды POP PSW

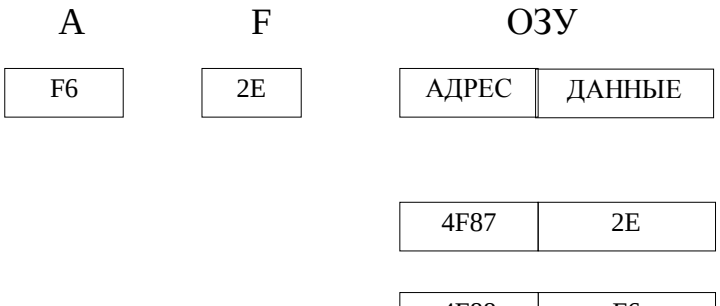


Рисунок 4. – Состояние МП и ОЗУ после выполнения команды POP PSW

Как видно из рисунка 3 и рисунка 4 работа со стеком при чтении напоминает извлечение пар листов из пачки. При этом листы отбираются с вершины пачки, а указатель стека хранит номер верхнего листа, оставшегося в пачке.

Совместный анализ рисунка 1, 2 и рисунка 3, 4 показывает, что работа на стеке осуществляется по принципу "первым вошел – последним вышел", т.е. данные, которые при последовательном обращении к стеку записывались в него первыми, будут прочитаны последними.

Порядок выполнения опыта

1. Ввести программу чтения из стека:

1000	C1H	POP BC
1001	D1H	POP DE
1002	E1H	POP HL
1003	F1H	POP PSW

2. Установить содержимое программного счетчика:

(PC) = 000H

3. Записать в ячейки ОЗУ с адресами 100H...108H такие данные, чтобы при чтении их в регистрах МП программой п.1 результат оказался бы равным (A) = 00H, (B) = 1H, (C) = 2H, (D) = 3H, (E) = 4H, (F) = 5H, (H) = 6H, (L) = 7H;

4. Выполнить программу п.1 и убедиться, что в регистрах МП находятся данные, оговоренные в п.3.

Контрольные вопросы

1. Что такое стек?

2. В какой области ОЗУ может быть организован стек?

3. Каким образом может быть классифицирована стековая память с точки зрения доступа к ней?

4. Каковы принципы записи в стек и чтение из стека в системе команд Intel 8080?

5. Какой регистр МП хранит адрес "верхушки" стека?

6. Каким образом можно передавать содержимое регистров DE в регистр указателя стека SP?

Лабораторная работа №4

МОДЕЛИРОВАНИЕ ПРОЦЕДУР ВВОДА ИНФОРМАЦИИ В МИКРО-ЭВМ

Цель работы: приобретение практических навыков моделирования с использованием средствIS580 процедур ввода квантованной информации.

Ввод информации в микро-ЭВМ по каналам является одной из основных задач при построении информационно-управляющих систем. Важность этой задачи обусловлена тем, что от организации ввода в значительной мере зависят такие характеристики измерительных каналов, как частота дискретизации, динамическая погрешность. Сопутствующими для ввода задачами, как правило, оказываются задачи первичной обработки: преобразование форматов данных, определение знака входных данных, линеаризации характеристик первичных измерительных преобразователей, нормирование, функциональное преобразование и т.д.

Грамотное решение задач ввода и первичной обработки, порой позволяет получить лучшие результаты на "медленных" АЦП и процессорах, чем неграмотное решение на "быстрых" АЦП и процессорах.

Задачи ввода распадаются на два относительно самостоятельных класса по типу дискретных сигналов: одноканальные бинарные и многоканальные бинарные. Значительно реже встречается задача ввода сигналов от трехпозиционных датчиков (по типу полярного реле). Одноканальные бинарные сигналы (статические и динамические)

характерны для задач программного логического управления, обработки информации импульсных датчиков (с частотной, фазовой и временной модуляцией). Многоканальные бинарные сигналы характерны для задач измерения и управления при наличии датчиков с непрерывным (аналоговым) выходом и подразумевают предварительные (до ввода) преобразования непрерывных сигналов в бинарные с помощью АЦП.

Опыт 1. Ввод одноканальных бинарных сигналов

Бинарный одноканальный сигнал, показанный на рисунке 1, может нести различную информацию.

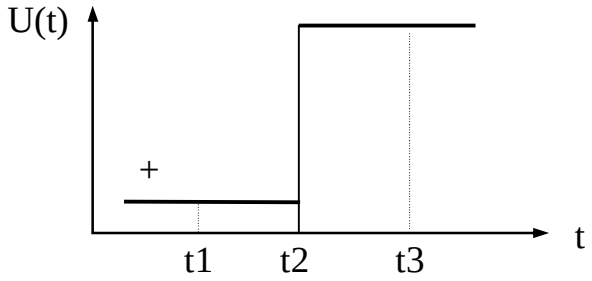


Рисунок 1 – Бинарный одноканальный сигнал

В статическом состоянии сигнала информацию несет состояние U(t) (логический "0" или логическая "1") в определенный момент времени (момент опроса). Так, при опросе датчика в момент времени t1 значение введенного байта будет равно "0", а при опросе в момент t3 – равно "1".

В данном случае информацию несет смена состояния бинарного сигнала с "0" на "1" (или наоборот). Здесь речь фактически о моменте t2 (рисунок 1).

Ввод бинарных сигналов в Intel 8080 может быть осуществлен единственной командой IN. При этом для одноканальных сигналов, как правило, приходится выполнять либо выделение нужного бита (например, соответствующим числом сдвигов с итоговым размещением вводимого сигнала в признак CARRY), либо маскирование ненужных битов (например, логическим умножением на байт, у которого во всех битах кроме интересующего расположены нули). При вводе нескольких бинарных одноканальных сигналов с целью упрощения их последующей логической обработки целесообразно хранить их в виде байтов с информационным битом в одноименных позициях байтов (например, старший или младший).

Порядок выполнения опыта

1. Написать ассемблерную программу ввода бинарных одноканальных статических сигналов из портов с адресами 01, 02, 03 и их логической обработки. Расположение битов в байтах и требуемую логическую функцию задает преподаватель в соответствии с таблицей 1;

2. Ввести программу, отладить ее, результат показать преподавателю;

Таблица 1 –Варианты заданий расположения битов в портах

Вариант №	Положение бита в портах			Логическая функция
	Порт 01	Порт 02	Порт 03	Y = F(x)
1 2	0 2	1 3	2 6	Y = X1^X2^X3
3 4	7 5	2 1	4 3	Y = X1vX2vX3
5 6	2 1	7 3	1 5	Y = X1^X2^X3
7 8	2 7	0 4	6 3	Y = X1vX2vX3
9 10	6 3	4 4	2 5	Y = (X1^X2)vX3
11 12	4 0	5 2	6 4	Y = (X1vX2)^X3
13 14	1 3	2 0	0 1	Y = X1+X2+X3
15 16	6 7	5 6	1 0	——————— Y = X1+X2+X3

3. Написать программу ввода динамического одноканального бинарного сигнала в соответствии с алгоритмом, показанным на рисунке 2. Вариант задания в соответствии с таблицей 2 получить у преподавателя;

4. Ввести программу, отладить ее, результат показать преподавателю;

5. Оценить максимальную погрешность при вводе динамического бинарного сигнала.

Таблица 2 – варианты заданий для программы

Вариант №	Адрес порта	Позиция бита	Тип перехода
1	1	0	“0” – “1”
2	2	1	“1” – “0”
3	3	2	“0” – “1”

4	1	3	“1” – “0”
5	2	4	“0” – “1”
6	3	5	“1” – “0”
7	1	6	“0” – “1”
8	2	7	“1” – “0”

Опыт 2. Ввод многоканального бинарного сигнала АЦП

Большинство АЦП имеют переменное время преобразования, зависящее от значения преобразуемого сигнала. В этой связи можно показать два возможных алгоритма ввода данных от АЦП. Синхронный алгоритм, показан на рисунке 3, предполагает, что максимальное время преобразования АЦП известно.

Алгоритм на рисунке 3 требует некоторых комментариев. Во-первых, расшифруем понятие "подготовка АЦП к преобразованию". С точки зрения выполняемых при этом функций логично отметить такие:

– сброс (установка) элементов памяти АЦП в исходное состояние;

– выбор канала (для многоканальных АЦП);

– настройка АЦП (выбор частоты квантования, опорных напряжений и т.д.).

Во-вторых, отметим, что временная задержка в синхронном алгоритме (рисунок 3) должна быть больше максимального времени преобразования АЦП. Наконец, сам термин "синхронный" обозначает, что данные вводятся без предварительного опроса готовности АЦП к обмену. Здесь речь по существу идет о равномерной дискретизации непрерывного сигнала во времени, так как алгоритм (рисунок 3) не имеет ветвлений.

Асинхронный алгоритм, показанный на рисунке 4, отличается наличием блоков определения готовности АЦП к обмену. Комментарии по организации асинхронного обмена можно найти в [1]. Здесь же отметим, что необходимость в определении готовности приводит к переменному интервалу ввода данных и делает дискретную непрерывную сигнала неравномерной во времени.

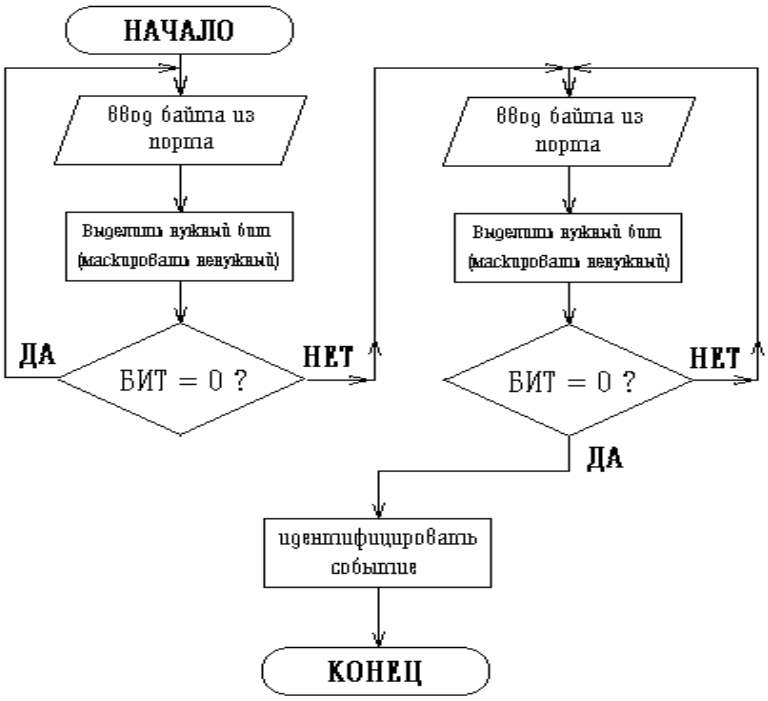


Рисунок 2 – Алгоритм ввода перехода “1” – “0”

Порядок выполнения опыта

1. Написать ассемблерную программу, реализующую синхронный алгоритм (рисунок 3). Данные о программе получить от преподавателя в соответствии с таблицей 3;

2. Временную задержку оформить подпрограммой. Для создания задержки использовать в цикле "пустые" операции (NOP; MOV R1, R1 и т.д.);

3. Ввести программу, отладить ее, результат показать преподавателю;

4. Модифицировать предыдущую программу в соответствии с асинхронным алгоритмом (рисунок 4). Дополнительные параметры программы получить у преподавателя в соответствии с таблицей 4;

5. Ввести программу, отладить ее, результат показать преподавателю.

Рисунок 3 – Синхронный алгоритм ввода данных от АЦП

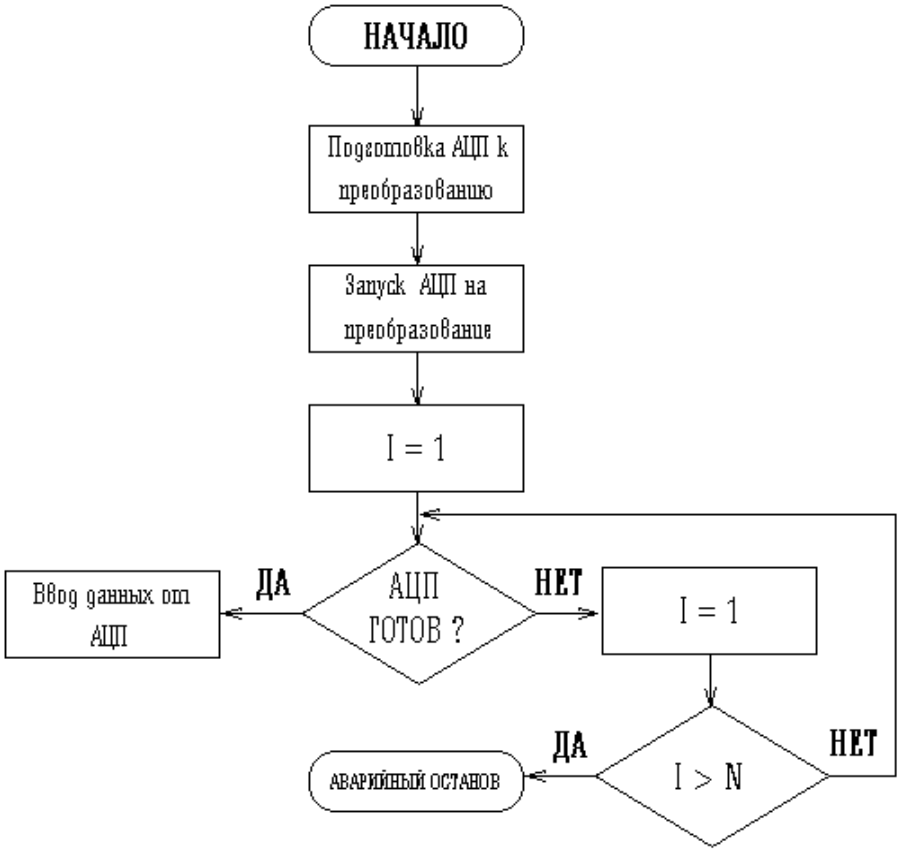


Рисунок 4 – Асинхронный алгоритм ввода данных от АЦП

Таблица 3 – Данные о программе

№ варианта	Величина задержки, с	Порт данных АЦП	Порт/байт подготовки АЦП	Порт/бит запуска АЦП
1	10	порт 01	02 - 04H	03 / X0 = 1
2	20	порт 02	03 - 08H	01 / X1 = 0
3	30	порт 03	01 - C0H	02 / X2 = 1
4	40	порт 01	02 - C4H	03 / X4 = 1
5	50	порт 02	03 - C8H	01 / X4 = 1
6	60	порт 03	01 - CCH	02 / X5 = 0
7	70	порт 01	02 - D0H	03 / X6 = 0
8	80	порт 02	03 - D4H	01 / X7 = 0

Таблица 4 – Дополнительные параметры прграммы

№ варианта	Значение N(10)	Порт/бит готовности
1	10	03 / X7 = 1
2	20	01 / X6 = 0
3	30	02 / X5 = 0
4	40	03 / X4 = 0
5	50	01 / X3 = 0
6	60	02 / X2 = 0
7	70	03 / X1 = 1
8	80	01 / X2 = 0

Контрольные вопросы

1. Каковы источники погрешностей при вводе одноканальных статических и динамических бинарных сигналов?

2. Каковы принципиальные пути снижения погрешностей ввода одноканальных бинарных сигналов?

3. Каковы типичные технические задачи, требующие ввода бинарных одноканальных сигналов?

4. Каковы сравнительные преимущества и недостатки синхронных и асинхронных алгоритмов?

5. Чем определяется величина задержки в синхронном алгоритме?

6. Чем определяется параметр N в асинхронном алгоритме?

7. В чем состоят процедуры подготовки, пуска и ввода из АЦП?

Список литературы

1. Методические указания к лабораторным работам по курсу "Вычислительная техника в измерительных устройствах и микропроцессоры".

Лабораторная работа №5

ОБРАБОТКА МАССИВОВ ИНФОРМАЦИИ В МИКРО-ЭВМ

Цель работы: изучить блок-схемы обработки массивов числовой информации в микро-ЭВМ, приобрести практические навыки в использовании рациональных методов адресации при работе с массивами, рассмотреть типовые для информационно-измерительной техники примеры обработки массивов.

Опыт 1. Изучение простейших блок-схем обработки массивов числовой информации

В информационно-измерительной аппаратуре с применением микропроцессоров и микро-ЭВМ достаточно часто встречаются задачи обработки массивов измерительной информации. При этом в общем случае предполагается, что К-мерный массив уже введен тем или иным способом в ОЗУ и необходимо осуществить его обработку по определенным алгоритмам. Такими задачами, например, могут быть: изменение знака чисел в одномерном массиве на противоположный (преобразование в дополнительный код), сортировка чисел по знаку и (или) по модулю, вычислительные операции с многомерными массивами (косвенные или совокупные измерения, например, косвенное определение мощности на постоянном или переменном токе), переформатирование (округление или усечении) данных и т.д. Характерной особенностью рассматриваемых задач является цикловая структура реализующих алгоритмов, причем для

многомерных массивов зачастую присутствуют и вложенные циклы (цикл в цикле). Для одномерных массивов достаточно общей структуры обработки, которая иллюстрируется на рисунке 1.

Особенностью, непопадающей под обобщенный алгоритм (рисунок 1), является необходимость сохранить исходные данные. В терминах вычислительной техники эта ситуация требует раздельного хранения исходного и обрабатываемого массива в ОЗУ. Для этого случая блок-схема (рисунок 1) может быть модифицирована к виду, показанному на рисунке2.

Порядок выполнения опыта

1. Получить от преподавателя вариант задачи по обработке одномерного массива в соответствии с таблицей 1;
2. Разработать и оттранслировать ассемблерную программу;
3. Включить рабочее место и ввести программу;
4. Отладить программу;
5. Результаты обсудить с преподавателем.

Опыт 2. Изучение особых случаев обработки массивов

Оба приведенных в опыте 1 алгоритма предполагают, что длина массива в результате обработки сохраняется, т.е. другими словами один исходный массив превращается в другой преобразованный массив той же длины. Однако существуют и задачи, которые по своей постановке требуют либо изменения (как правило, сокращение) длины массива, либо изменения количества массивов. Примером сокращения длины массива при обработке может служить задача выборки положительных (отрицательных) чисел из исходного массива.

Примером изменения количества массивов может служить задача сортировки чисел на положительные и отрицательные (один исходный массив превращается в два) и задача вычисления суммы элементов массивов (два исходных массива превращаются в один).

Таблица 1 – Варианты задачи обработки одномерного массива

№ варианта	Необходи-мость	Начальный адрес	Длина массив	Начальный адрес	Сущность алгоритма обработки

	в сохранении исходного массива	исходного массива	а в байтах	обрабатывая-емого массива	
1	нет	4000H	28H	-	Изменить знак элемента массива на противоположный
2	да	3000H	16H	2000H	—— ‖ ——
3	нет	3600H	18H	-	Добавить к элементу массива (байта) аддитивное смещение 02H. При переполнении разрядной сетки результату присваивают значение FF¦
4	да	2750H	44H	1600H	—— ‖ ——
5	нет	1260H	12H	-	Сдвинуть байт на 2 бита вправо (разделить байт на 4)
6	да	3780H	32H	2400H	—— ‖ ——
7	нет	5100H	20H	-	Замаскировать старший бит
8	да	4150H	8H	1210H	—— ‖ ——
9	нет	6540H	10H	-	обнулить младшую тетраду байта
10	да	1700H	24H	5000H	—— ‖ ——

Порядок выполнения опыта

1. Получить от преподавателя вариант задачи в соответствии с таблицей 2;

2. Адреса и длины массивов выбрать самостоятельно;

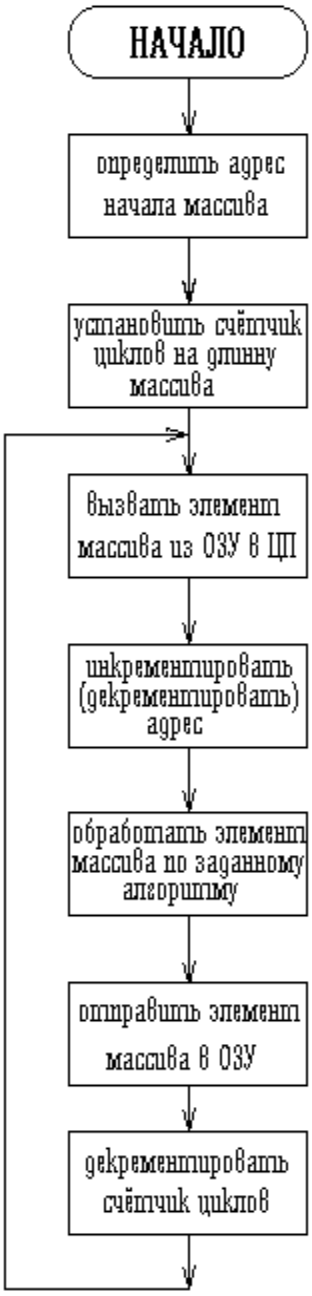


Рисунок 1 – Обобщённая блок-схема обработки одномерного массива

Таблица 2 – Варианты задач обработки массива с сохранением данных

№ варианта	Формулировка задач
1.	Из исходного массива отобрать положительные числа исходный массив сохранить в ОЗУ
2.	Из исходного массива отобрать отрицательные числа. Новый массив разместить, начиная с начального адреса исходного массива
3.	Рассортировать исходный массив по знаковому признаку
4.	Рассортировать исходный массив по признаку четности числа единиц в байте
5.	Сложить элементы двух исходных массивов. Переносами пренебречь
6.	Вычислить поразрядную конъюнкцию элементов двух массивов

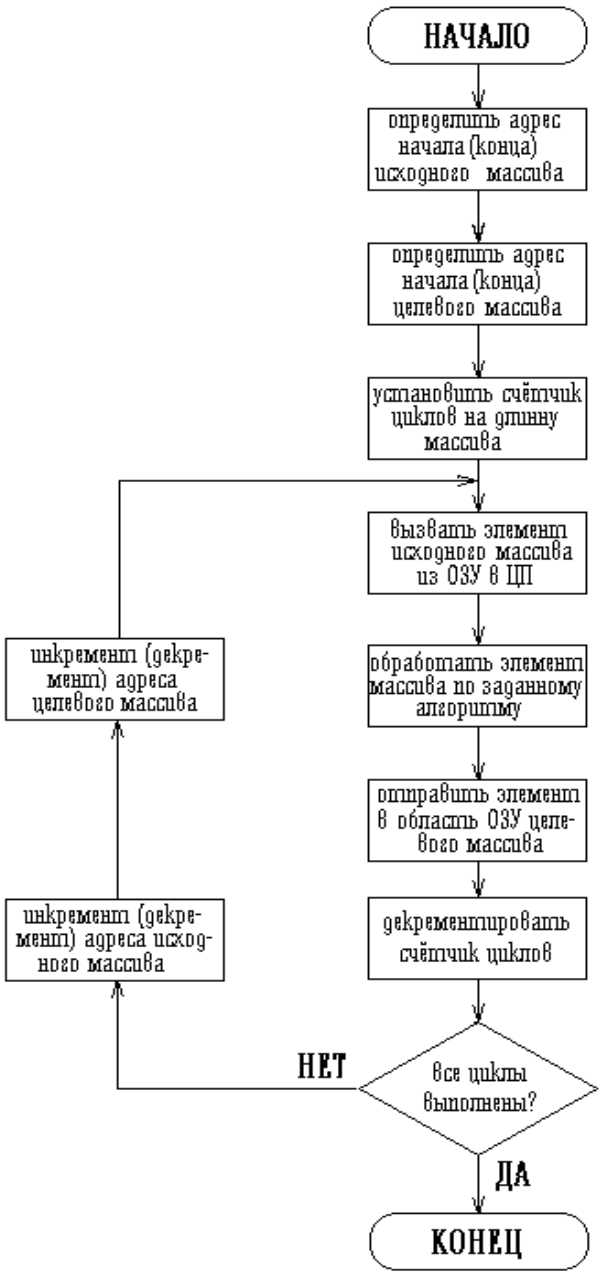


Рисунок 2 – Блок-схема обработки одномерного массива для случая, когда необходимо сохранение исходного массива

Содержание отчета

В отчете о лабораторной работе необходимо привести блок-схемы и ассемблерные программы выполненных задач.

Контрольные вопросы

1. Какова характерная особенность структуры программы для обработки массивов?

2. Что такое длина и размерность массива?

3. Каковы типичные задачи, связанные с обработкой массивов?

4. Какие способы адресации выгодно использовать при обработке массивов?

5. Каковы качественные состояния между количеством и длиной исходных и обработанных массивов?

6. Каким образом при обработке массивов можно использовать индексную и относительную адресацию?

7. Чем ограничены размеры обрабатываемых массивов?

8. Каким образом необходимо обрабатывать массивы, элементы которых имеют большую длину, чем разрядная сетка процессора?

Лабораторная работа №6

ПРИМЕНЕНИЕ МИКРО-ЭВМ ДЛЯ МОДЕЛИРОВАНИЯ АППАРАТНЫХ СРЕДСТВ

Цель работы: овладение приемами создания программных моделей аппаратных средств (на примерах комбинационных логических устройств).

Необходимость в создании программных моделей аппаратных средств возникает либо в ходе проектирования аппаратного обеспечения, либо при перераспределении функций между аппаратными и программными средствами системы. Примером первой из указанных ситуаций может быть создание некоторой логической схемы. Для проверки правильности ее функционирования, выяснения возможного наличия "гонок" при прохождении сигналов и окончательной отладки вовсе необязательно собирать макет данной схемы. Бывает целесообразным (особенно при наличии банка моделей) решить эти проблемы путем моделирования. Примером второй из указанных ситуаций является возложение на программную часть некоторой системы функций аппаратной части с целью минимизации последней. Такими функциями, в частности, могут быть: преобразование кодов, счет, формирование временных интервалов и задержки, формирование регулярных и нерегулярных импульсных последовательностей и т.д.

Таким образом, программные модели аппаратных средств являются достаточно мощным средством проектирования устройств и систем. Их использование может значительно сократить время разработки, отладки, тестирования и модификации устройств и систем, сократить аппаратные затраты при одновременном обеспечении гибкости средств, быстро и высокоэффективно исследовать все возможные режимы работы. Различают два способа построения модели: компиляционный и интерпретирующий.

При компиляционном способе для каждого объекта моделирования конструируется своя программа, которая для выполнения не требует никаких входных данных, кроме входного набора переменных (входного вектора).

При интерпретирующем способе конструируется достаточно универсальная программа, моделирующая работу группы или даже класса объектов. Работа с такой моделью, помимо входного вектора переменных требует и задания вектора настройки, т.е. некоторого массива данных, по которым осуществляется настройка программы на конкретный тип модели.

В данной работе изучаются приемы моделирования комбинационных логических устройств (схем без памяти).

Опыт 1. Моделирование интегральной схемы К155ЛР1

Интегральная схема К155ЛР1, как известно, реализует следующую функцию алгебры логики:

$$Y_1 = \overline{X_1 \wedge X_2 \wedge X_3 \wedge X_4} \qquad (1)$$

$$Y_2 = \overline{X_5 \wedge X_6 \wedge X_7 \wedge X_8}\ , \qquad (2)$$

где Y_i – входные переменные; i $\in$ 1...8;

Y_j – выходные переменные; j $\in$ 1,2.

Рассмотрим алгоритм моделирования функций (1), (2).

Пусть в некоторый регистр R1 процессора тем или иным образом введен входной вектор

$$(R_1) = X_8 X_7 X_6 X_5 X_4 X_3 X_2 X_1$$

Сдвинем этот вектор циклически вправо и разместим результат в регистре R2

$$(R_2) = X_8 X_1 X_7 X_6 X_5 X_4 X_3 X_2$$

Результат поразрядного логического И над содержимым регистров R1 и R2 разместим в регистре R3

$$(R_3) = (X_8 \wedge X_1)(X_7 \wedge X_8)(X_6 \wedge X_7)(X_5 \wedge X_6)(X_4 \wedge X_5)(X_3 \wedge X_4)(X_2 \wedge X_3)(X_2 \wedge X_1)$$

Содержимое R3 сдвинем циклически вправо на два бита и результат разместим в регистре R4

$$(R_4) = (X_2 \wedge X_3)(X_1 \wedge X_2)(X_8 \wedge X_1)(X_7 \wedge X_8)(X_6 \wedge X_7)(X_5 \wedge X_6)(X_4 \wedge X_5)(X_3 \wedge X_4)$$

Поразрядное логическое умножение содержимого регистров R3 иR4 даст

$$(R_5) = \left[(X_8 \wedge X_1) \vee (X_2 \wedge X_3)\right]\left[(X_7 \wedge X_8) \vee (X_1 \wedge X_2)\right]\left[(X_6 \wedge X_7) \vee (X_6 \wedge X_1)\right]$$
$$\left[(X_5 \wedge X_6) \vee (X_7 \wedge X_8)\right]\left[(X_4 \wedge X_5) \vee (X_6 \wedge X_7)\right]\left[(X_3 \wedge X_4) \vee (X_5 \wedge X_6)\right]$$
$$\left[(X_2 \wedge X_3) \vee (X_4 \wedge X_5)\right]\left[(X_1 \wedge X_2) \vee (X_3 \wedge X_4)\right]$$

Нетрудно видеть, что инверсия содержимого регистра R5 даст искомый результат в младших битах тетрад (символом "*" помечены не интересующие нас биты)

$$(R_5) = ***\overline{(X_8 \wedge X_7 \wedge X_6 \wedge X_5)} ***\overline{(X_1 \wedge X_2 \wedge X_3 \wedge X_4)}$$

Изложенное позволяет представить блок-схему алгоритма моделирования в виде рисунка 1.

Порядок выполнения опыта

1. Определить в качестве R1..R2 (рисунок 1) конкретные регистры процессора;

2. Написать ассемблерную программу, реализующую алгоритм (рисунок 1);

3. Ввести программу в ОЗУ и отладить ее; получить от преподавателя задание на входной вектор X, выполнить моделирование и результат показать преподавателю.

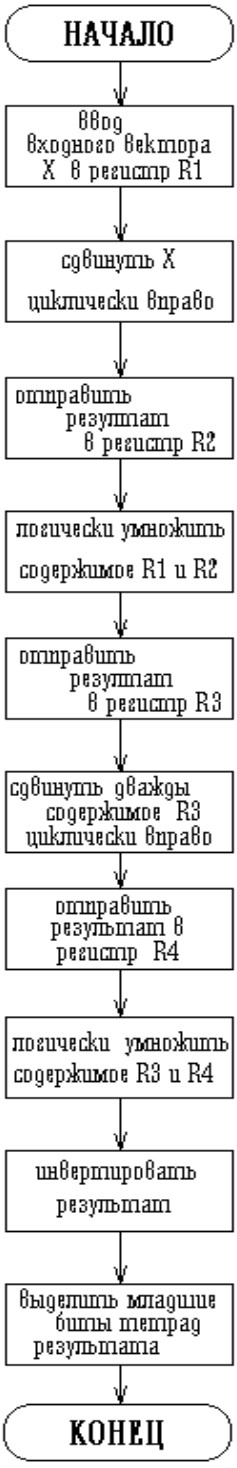


Рисунок 1 – Блок-схема алгоритма моделирования

Опыт 2. Моделирование преобразователя кодов

Программное моделирование ряда логических устройств может быть осуществлено без вычислений функций алгебры логики, как это делалось в опыте 1. В частности, это справедливо для преобразователей кодов, программные модели которых проще всего создавать

табличными методами. Рассмотрим алгоритм моделирования преобразователя двоичного кода 8-4-2-1 в двоично-десятичный 8-4-2-1. Положим, что входная информация - беззнаковая и имеет длину слова в один байт. Особенностью рассматриваемой задачи является то, что входные числа большие, чем 9910, но меньшие, чем 25610 должны представляться в двоично-десятичном коде тремя тетрадами, т.е. иметь длину в полтора байта. Поэтому, для унификации процедур выборки из таблицы целесообразно выбрать длину выходного слова преобразователя в два байта. При этом нулевые значения соответствующих тетрад в зависимости от входной информации должны соответствовать рисунку 2.

старший байт		младший байт		входное число(10)
0	0	0	0	0
0	0	0	*	1…9
0	0	*	*	10…99
0	*	*	*	100…255

Рисунок 2 – Кодирование двоично-десятичных чисел в двухбайтном формате

Порядок выполнения опыта

1. Получить от преподавателя задание на диапазон преобразуемых кодов;

2. Создать в соответствующих областях ОЗУ первую и вторую части таблицы;

3. Написать ассемблерную программу;

4. Ввести программу в ОЗУ. Обратить внимание на необходимость ее размещения вне области 0100...02FF (занято таблицей);

5. Отладить программу, выполнить моделирование и результат показать преподавателю.

Таблица 1 – Расположение в ОЗУ двоично-десятичных кодов

Адрес (16)	Данные(16)	Адрес (16)	Данные(16)
младшие байты		старшие байты	
0100	00	0200	00
0101	01	0201	00
0102	02	:	:
:	:	0262	00
0109	09	0263	00
010A	10	0264	01
010B	11	0265	01
:	:	:	:
0162	98	02C7	01
0163	99	02C8	02
0164	00	:	:
0165	01	02CE	02
:	:	021F	02
01C6	98		
01C7	99		
01C8	00		
01C9	01		
:	:		
01FD	53	3	
01FE	54		
01FF	55		

Опыт 3. Моделирование мультиплексора К155КП5

Алгоритм работы мультиплексора К155КП5 сводится к коммутации с инвертированием одного из входов X0...X7 на выход Y в зависимости от состояния управляющих входов S0...S2. Эту модель также можно создать без непосредственного вычисления функций алгебры логики. Сущность одного из возможных алгоритмов решения рассматриваемой задачи состоит в следующем. Осуществляется циклическое сравнение вектора $S \in 0...7$ и декрементируемого счетчика циклов (СЦ). При неравенстве каждый раз происходит сдвиг вектора X влево. В момент равенства (СЦ) = S происходит выход из цикла. При этом очевидно, что нужный бит входного вектора X находится в результате сдвигов в позиции старшего бита.

Инвертирование и выделение этого бита приводит к окончательному результату. Выделение требуемого бита можно осуществить, например,

выдвижением его во флаг CARRY. Соответствующая описанному алгоритму блок-схема приведена на рисунке 4.

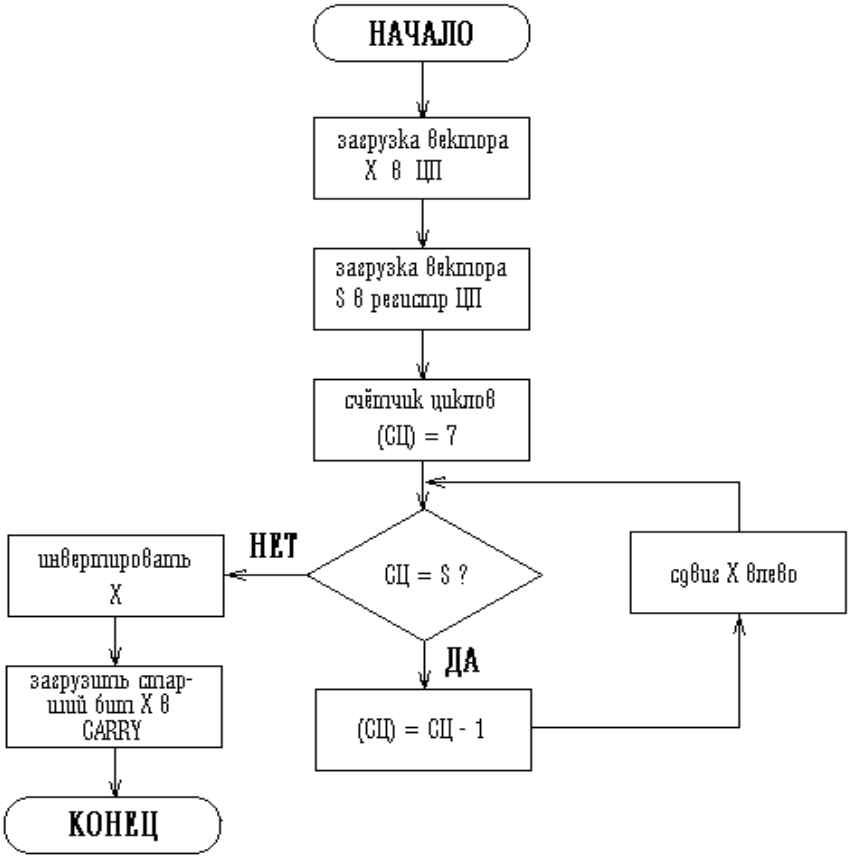


Рисунок 4 – Блок-схема алгоритма работы мультиплексора

Порядок выполнения опыта

1. Написать ассемблерную программу, реализующую алгоритм (рисунок 4);

2. Ввести программу в ОЗУ;

3. Отладить программу, выполнить моделирование, результат показать преподавателю.

Контрольные вопросы

1. Для чего используются программные модели аппаратных средств?

2. Каковы способы построения программной модели?

3. Как модифицировать алгоритм (рисунок 1) в целях автоматизации моделирования при любом значении вектора X?

4. Как оптимизировать алгоритм (рисунок 3) по объему табличной памяти?

5. Как можно представить алгоритм, альтернативный алгоритму (рисунок 3)?

6. Как из компиляционного алгоритма (рисунок 4) сделать интерпретирующий (для разных типов мультиплексоров)?

Список литературы

1. Овчаренко А.И., Программирование на ассемблере: Уч. пособие. – К.: ИСИ, 1993. -135 с.

Навчальне видання

«Програмування на асемблері».
Методичні вказівки до лабораторних робіт
з курсу «Основи мікропроцесорної техніки»
для студентів спеціальностей 7.091301
“Інформаційно-вимірювальні системи”, 7.091302
“Метрологія та вимірювальна техніка”
денної та заочної форм навчання
Російською мовою

Укладачі: ОВЧАРЕНКО Олександр Іванович,
ЛИСЕНКО Володимир Валерійович,
МИГУЩЕНКО Руслан Павлович,
МЕДВЕДЄВА Людмила Олександрівна,
КРОПАЧЕК Ольга Юріївна,
ШАПІРО Михайло Віталійович

Відповідальний за випуск В.І. Дякін
Роботу до друку рекомендував О.І. Рогачов

В авторській редакції

План 2007р., п. 181
Підписано до друку . Формат 60×84 1/16. Папір офсет. Друк – ризографія. Гарнітура Таймс. Умов. друк. арк. 3.2. Облік. – вид. арк. 3.5. Наклад 100 прим. Ціна договірна.

Видавничий центр НТУ “ХПІ”, 61002, Харків, вул. Фрунзе, 21
Свідоцтво про державну реєстрацію ДК №196 від 10.07.2000р.

Друкарня НТУ “ХПІ”